Handling Environment for Publicly Posted Composite Documents

Helen Balinsky, David Subirós Pérez

Handling Environment for Publicly Posted Composite Documents

Helen Balinsky, David Subirós Pérez

Hewlett-Packard Laboratories
Long Down Avenue
Bristol, UK
{helen.balinsky, david.perez5}@hp.com

Abstract. Recent business needs and requirements for complex cross-organizational workflows led to development of Publicly Posted Composite Documents (PPCD) – a multi-part document format retaining parts in their original formatting for usability, user convenience and information availability, while providing document security and built-in access control for inter- and intra-organizational document workflows distributed over low security channels. Developing PPCD Handling Environments – Authoring and Workflow Participant Access – for creating and accessing PPCD documents posed many new challenges: composition retention, secure access and management of multi-formatted, encrypted data on laptops and potentially low capability devices. The paper describes the use of OLE Automation, Add-Ons and System Call Interception technique to preserve the overall document composition, while the handling of individual parts is delegated to external tools, native for individual document formats.

Keywords: Publicly Posted Composite Documents · cross-organizational workflow · OLE Automation · Add-Ons · System Call Interception · security · integrity · authenticity · availability · authoring environment

1 Introduction

In recent years the complexity and distribution of business collaborations has dramatically increased from merely crossing organizational boundaries to crossing countries and continents, where the organizations involved share no trusted infrastructure/s for document access control. This created an urgent need for self-contained customized digital document bundles, where all necessary information is securely assembled together to provide data and information availability at multiple distributed workflow decision points. Composite documents participating in these complex workflows are shared between different organizations over low security communication channels, such as public clouds or 3rd party servers.

Publicly Posted Composite Documents (PPCD) [1] addressed the need for secure composition by providing document format, capable of simultaneously containing multiple individual documents, such as Microsoft Word, Adobe Acrobat, html, xml or

adfa, p. 1, 2011.

any other required, whilst providing fast access to each individual content-part without the need to traverse the entire bundle. The rationale behind retaining native formats lies in the fact that each document format offers its unique benefits (e.g. ability to perform computations in Microsoft Excel, ease of creating presentations in Microsoft PowerPoint, convenience of certifying Adobe PDF), making different formats more suitable for diverse business needs and anticipated uses. Conversion of document parts from their native formats into a common format (e.g. Adobe pdf) will inevitably result in substantial drawbacks, such as limited editing capabilities, inability to perform in place computations and others.

This imposes hard challenges on both the Authoring and Participant Access of the PPCD handling environment (HE). A PPCD HE needs to seamlessly integrate and interact with a vast, potentially unrestricted number of document applications each of which can be used to format a document part. The problem of application interoperability is not new and is partially addressed by the latest versions of Internet Explorer and Microsoft Outlook, where some received attachments can be pre-viewed directly in the Outlook environment (Fig. 1, left), but this capability is limited to previewing of supported formats only (Fig. **1**, right).

PPCD HEs are taking these challenges much further. An individual document part needs to be extracted from a composite document, authenticated, decrypted, associated with its native application and then streamed to the application instance to be presented to a user according to the access granted: read only (RO) or read write (RW). Furthermore, RO mode requires a content-part to be opened in *View* (non-editable) mode with *Save* functionality disabled, which can be relatively easily enforced for some applications [2], whilst not for others. RW access requires a substantially more complicated interaction: a HE needs to acquire modified contents from an earlier activated native application, apply the required security protection and replace the data back into the original composite document. While delegating content handling to a native application is a relatively straightforward task, ensuring automatic streaming of modified contents back to the PPCD HE is a challenging task for most applications.

The other challenge is controlling *Save*/*SaveAs* functionality of native applications, where a content-part may be exported out of a PPCD HE, potentially preventing the latest modification from being incorporated back into the corresponding content-part. From a security prospective, this could be a potential channel for sensitive data leaks.

The paper is organized in the following way. The state of the art is provided in Section 2. In Section 3, for the convenience of the reader, we recapitulate the PPCD structure and access procedures, and then present the problem statement. The high-level overview, architecture and detailed description of the solution components are described in Section 5. Implementation details of the current prototype are described in Section 6. Finally, in Section 7 we draw our conclusions and discuss plans for future work.

2 State of the Art

The current paper is the first attempt to investigate the complexity and understand the challenges encountered by the Handling Environment of a PPCD-formatted composi-

tion, hence there is no direct prior art. Nevertheless, similar data security solutions [3] are developed using other paradigms that do not require dedicated handling environments. A good example is the encryption software by Symantec. Symantec File Share Encryption [3] is a file and folder encryption software with policy enforcement to enable team collaboration and secure file sharing through insecure channels (e.g. public servers or cloud storage). Symantec Desktop Email Encryption delivers client-based email encryption that automatically encrypts and decrypts emails as they are sent and received on desktops and laptops, providing protection for end-to-end communications. Nevertheless, it does not provide data-centric protection; when a document leaves the email system, the data is no longer protected. The solution integrates with existing e-mail clients and thus does not require a dedicated handling environment. In addition, neither solution provides differentiable access control for individual documents nor the ability to combine them into a coherent composition.

3 PPCD Structure and Access Procedures

In this section, we briefly recall the main structure and key components of PPCD; more details can be found in [1], [4-6]. A PPCD document is a multi-part composition of individually accessible content-parts (schematically illustrated in Fig. 2, left).

Each content-part can be a traditional document (e.g. *.doc/x, *.ppt/x, *.pdf, etc.) or a combined group of documents that require the same access for the duration of a workflow - simultaneously accessible, or not, by every participant in the workflow.

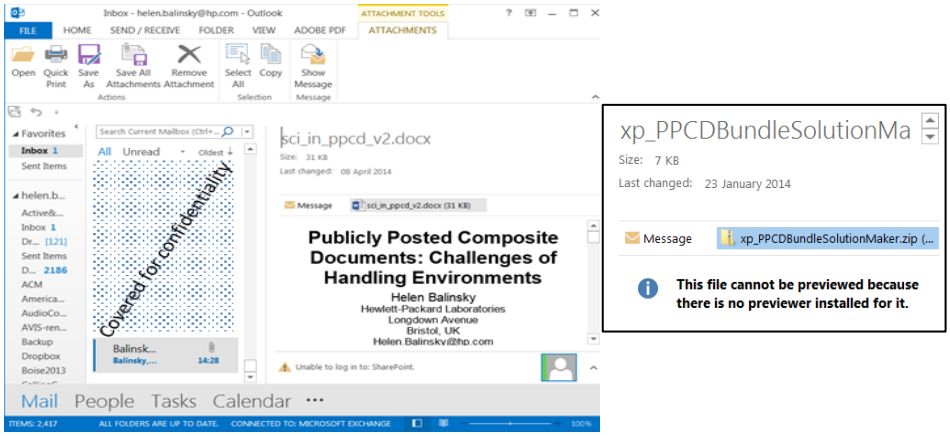

Fig. 1. (left) Embedded view (no editing) of a Microsoft Word document in Outlook. (right) Embedded view unavailable for an application.

Each content-part at every workflow step can be granted one of three types of accesses: read write (RW), read only (RO) or validate authenticity (VA). Each participant, when required to access a PPCD-formatted document, is provided with a corresponding subset of keys to each part according to access granted (RO/RW). In complex workflows a participant with higher granted access may be preceded by a participant

with lower access (e.g. a contributor is followed by a decision maker), thus the lower access participant may be required to handle "inaccessible" parts – without RW/RO access granted.

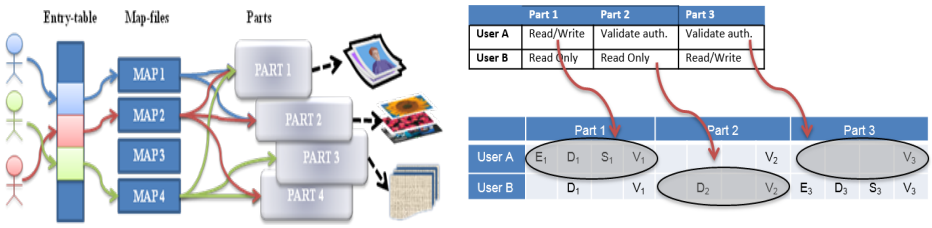

Fig. 2. (left) Schematic diagram of PPCD structures. (right) Each participant is given corresponding signature verification keys for every part and access keys for parts with RO/RW access only.

To ensure that every part with or without RW/RO access remains authentic while it is transitioned over lower security channels or through users with lesser privileges, each content-part is digitally signed after being encrypted by a signature key assigned to the content-part. Each workflow participant is provided with the corresponding signature verification key for each content-part irrespective of access rights granted. RO access to a part is provided through a part decryption key, whilst RW – through decryption/encryption and signature keys. Thus, a participant with RO access to a content-part can decrypt it for reading, but cannot authorize (sign) any modifications made to the content-part, and only a participant with RW access to the content-part is provided with the corresponding signature key. An example of key distribution for a workflow with 2 users with 3 content-parts is illustrated in Fig. 2, right. All PPCD content-parts and key-maps are assembled in SQLite [7] flat file database and accessible individually according to the granted access. Unlike other serialization techniques, SQLite provides fast (non-sequential) access to each individual component.

4 Problem Statement

This paper addresses the problems and challenges of developing PPCD Handling Environments (HEs): namely the Authoring Environment and the Workflow Participant Access. Both types of PPCD HEs operate similarly and provide access to PPCD documents, however only the Authoring Environment provides functionality for creating new PPCD-formatted documents and workflows. In this section, we describe the challenges of creating PPCD HEs, which will be addressed in Section 5, and implementation details provided in Section 6.

Once the authenticity of a PPCD document is established ([1], [4-6] for more details), individual content-parts are decrypted ready to be displayed. The native applications are identified using the part name extension (and/or the file beginning), however, most document-handling applications do not accept streaming data (in our case from PPCD content-part decryption module) and require hard drive files to work with.

Then the corresponding access rights RO/RW need to be enforced, functionality that is conveniently supported by some native applications, but unfortunately not by others.

The next challenge is to safe-handle potentially sensitive content-parts whilst temporarily storing them on a hard drive and delegating their management to the corresponding native application. One possible mechanism is to run a native application in an embedded mode within PPCD HE. This could also provide for homogeneous access and handling of content-parts from the PPCD HE. However, embedded mode is only supported by a very limited number of document handling applications, thus calling an assigned application for handling a particular format in a stand-alone mode becomes unavoidable. This poses even more challenges:

- Retrieving updated contents of a RW content-part after it has been modified by a native application running outside PPCD HE framework.
- Removing all temporary copies of a sensitive content-part created by a native application.
- Ensuring that sensitive content-parts cannot escape PPCD HE to prevent potential sensitive data leaks, e. g preventing content-parts from being exported to any remote/removable media.
- Guaranteeing the integrity of a PPCD composition: any authorized edit should be incorporated to the corresponding PPCD composition.

5 Our Solution

In this section, we describe the high-level overview and key components of the proposed architecture for a PPCD HE. Fig. 3 illustrates the overall view on PPCD HE architecture. More in-depth details are provided in the following sections of this paper.

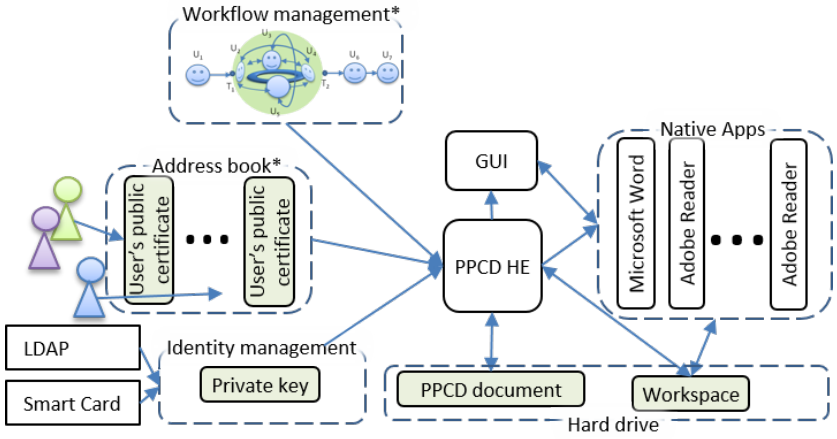

Fig. 3 The overall view of PPCD HE architecture

5.1 High Level Overview

The main components/modules of PPCD HE include:

- The module for accessing PPCD formatted compositions including authenticity validation, encryption/decryption services, serialization/deserialization (SQLite) access, and others as described in Section 3. The Authoring Environment (AE) also provides the key generation / derivation functionality required for the creation of new PPCD-formatted documents and document workflows.
- The overall application GUI, also providing an interface for native applications running in embedded mode (where applicable).
- Identity Management and Contact Book modules: either internal components or connectors to external applications and services (e.g. LDAP repositories, Cloud-based Identity services) for accessing private keys and public key certificates of document masters and workflow participants. The Contact Book module is only required for the AE, where workflow participants are assigned to a workflow by their public key certificates.
- Document workflow and access control module for determining workflow steps, workflow participants, their roles, access orders and required access rights. This could be an internal module (as described in Section 6) or a connector to an external business process management module, where workflow roles and required documents are established. This module is only provided for the AE.
- Native applications management module, identifying and managing native applications associated to content-parts.
- Session-based Workspace management module: providing and managing secure facilities for data exchange between the module for accessing PPCD-formatted compositions and native applications.
- The module for augmenting and controlling the behavior of native applications based on OLE Automation ([8], [9]), Add-Ons and System Call Interception (SCI) [10].

Data Layer: PPCD and Workspace. The data layer includes the elements that can be stored in a hard drive: a PPCD-formatted composition, which was described in Section 3, and the Workspace, which is used by the HE to temporarily store parts of a PPCD-formatted composition while it is being accessed. The Workspace is likely to be restricted to a local hard drive, while a PPCD document can be accessed from a local, remote or removable media, or from cloud storage.

When a user accesses a content-part with RO or RW rights, PPCD HE decrypts its contents into a Workspace. The clear text content is accessible only by the PPCD HE main process and native applications, which are run as children of the main process.

Native Application Module. Presented with a content-part, PPCD main application determines the corresponding native application/s using the extension of the part name and the file header. While the PPCD HE can use applications registered to each extension by the OS, it is likely to maintain its own list: PPCD HEs will require run-

ning slightly modified versions of native applications in order to preserve integrity of PPCD composition, while delegating handling of individual parts to their corresponding applications. PPCD HE requires integration with different native applications to provide reliable data exchange. In the current paper we present 3 mechanisms that can provide the required integration:

1. *OLE Automation* ([8], [9]) for applications supporting embedded mode: PPCD HE provides full control over input/output of an embedded application as well as control over enabled features and functionalities.
2. *Add-Ons* for applications that provide the required customization: PPCD HE executes an instance of a native application with pre-installed Add-Ons, which augment *Save*/*SaveAs* functionality according to the requirements of the PPCD HE. This is the preferred approach for applications supporting Add-Ons.
3. *The remaining applications* that can neither be embedded into PPCD HE, nor customized through Add-Ons: to control and augment *SaveAs* and other export functionality of these applications, the System Call Interception technique [10] is used. Native applications are injected with the behavior modifying code.

Further details of these mechanisms are provided in Section 5.3.

Identity Management and Contact Book. While the PPCD HE can potentially provide its own facilities to handle the master's private key, in real life deployments the document master key is likely to be stored in the certificate/key store provided by the OS, in user's Smartcard or Active Identity, centralized LDAP store (usually role-based key) or Cloud-based Identity services. Thus, the PPCD HE will require some secure connectors for accessing the user's/role's private keys.

To handle contact details (including public key certificates) of the current and potential workflow participants a Contact Book is required. It can either be provided by the PPCD AE or through its integration with a corporate LDAP or personal Contact Book. A snapshot of a simple LDAP-based Contact Book is shown in Fig. 7 left.

Workflow Management. As mentioned previously, the Workflow Management module is a part of the AE only. The minimalistic workflow management module will require a list of workflow participants (as determined in Section 5.1 above), and content-parts (placeholders for new parts to be added/created during a workflow). Then the document master determines the number of workflow steps required, the order of access by different workflow participants, and subsequently the access required by workflow participants at every work-flow step. The PPCD access module then automatically generates keys for all the content-parts (Fig. 4) and distributes them into the key-maps of participants according to access rights assigned.

More advanced PPCD AE is likely to integrate into business/process software (e.g. [11]), where a sequence of workflow steps can be automatically extracted, etc.

Based on the needs of business workflow a document master selects document parts required for a workflow, assigns individual participants or roles, determine the access order and required access to each content-part by each participant at every

workflow step. Many business workflows are very repetitive, so once a PPCD workflow template is created it can be reused as is or adjusted/modified as needed.

5.2 Creating and Accessing PPCD

Session Workspace. Most document-handling applications are not designed to accept data streams as their input and require a hard drive file to work with. To preserve the integrity of a PPCD all currently opened PPCD content-parts are localized into a dedicated Workspace (WS) folder, created at the beginning of each session: whether an existing document is opened or a new one is created. The opened WS is automatically removed when the PPCD HE session closes. The key challenges here are:

Problem 1. To contain a composite document within a PPCD HE, and its corresponding WS, so that none of the spanned standalone native applications (running independently from the PPCD HE) can scatter content-parts over local, remote and removable storage media.

Problem 2. To protect a WS from being accessed by any process that does not belong to the same session as the current WS.

Depending on the device capabilities and/or user preferences, a PPCD document can be accessed in 'eager' or 'lazy' modes, loading the contents immediately on document access or delaying the loading until explicitly required. The lazy mode is more suitable for low capability devices or bigger composite documents, whilst the eager mode provides faster access and jump between different content-parts

Opening a PPCD. As we mentioned in the previous section, when a PPCD-formatted document is opened, a new WS is created, whose location is a configurable parameter, denote it $WS_ID. PPCD document authentication is performed by a PPCD HE as described in [1]. Upon successful authentication, the HE performs the following steps (depicted in Fig. 4 left) for RO/RW content-parts

Step 1. Start a child process C dedicated to content-part handling.

Step 2. Retrieve the part name P and its contents from PPCD serialization. Determine a native application to be called using the part name extension and / or the file header.

Step 3. Create a new file P: $WS_ID\part name in the current WS.

Step 4. Lock the file P for an exclusive access by the process C.

Step 5. Write the contents of the decrypted content-part into the file P.

Step 6. Reopen the file P according to access granted to the user (RW or RO).

Step 7. Execute the native application, and if it supports View or Edit modes, select according to access granted.

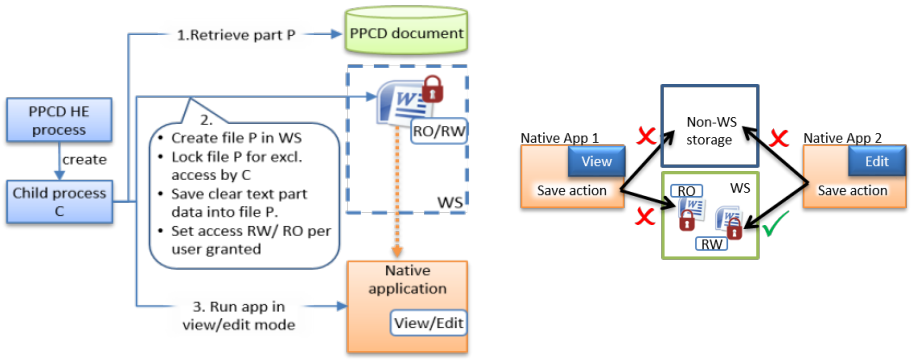

Fig. 4. (left) Opening of a PPCD document by a PPCD HE; (right) A native application 1 in a real or unsupported View mode cannot save modifications in either in WS or outside. A native application 2 in Edit mode can only save modifications to the corresponding WS.

An assigned native application can be called in embedded (if supported, [12]) or stand-alone mode to display the contents of a part. For any native application that supports separate View/Edit modes, a PPCD HE needs to load the application in the mode corresponding to access given. For example, Microsoft Word could be called in View mode or normal Edit mode, whilst pdf files can be opened with Adobe Acrobat for RW and Reader for RO access. In the cases where View mode is not supported by a native application, a separate warning is displayed notifying the workflow participant that modifications will not be incorporated into the composite document.

In order to guarantee exclusive access to files in a WS, the PPCD HE creates a child process *C* for every content-part, which decrypts the part and saves it into the current WS using the part name as the file name. *C* subsequently locks the file for exclusive access by itself and its children using, for example, *LockFile* [13] function from Windows API. This ensures that no other process can access the file. The native application, launched by *C,* subsequently reopens the file in the mode defined by the access granted.

Closing a PPCD. When a PPCD-formatted document is closed, all outstanding modifications for content-parts with RW access need to be incorporated. Modified content-parts in the WS are read, encrypted, signed by the corresponding keys and then placed into the PPCD document. All unauthorized modifications (if any) are automatically discarded and the WS is removed (possibly shredded with HP *FileSanitizer* [14] or similar applications for sensitive content-parts).

Fig. 4 right illustrates different behavior of native applications in View and Edit modes.

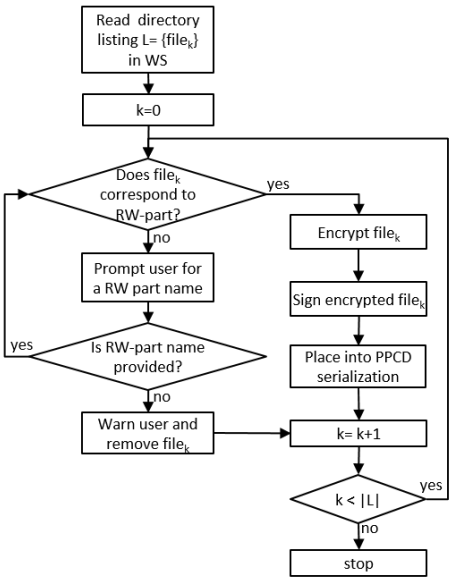

Fig. 5. Logic followed by PPCD HE to close a session.

A native application in View mode must not accept any modifications, while an application in Edit mode may only save modifications to the current WS. This modified behavior of native applications is achieved through OLE Automation, Add-Ons or SCI. We will describe the techniques in detail in Section 5.3.

To close a session or save current modifications, the PPCD HE follows the logic illustrated in Fig. 5. The list of files in the corresponding WS is retrieved. Every file name is checked against the corresponding part name in the opened PPCD composition. If the corresponding part is found with RW access granted, the part contents are encrypted and signed by the part assigned keys according to the PPCD scheme (see Section 3 and [1] for more details on keys used). If no part with the file name is found in the PPCD composition, the user may be prompted to specify the file name of the part he wants update. If a suitable part name is not provided or the modified file corresponds to a RO file in the WS, the modified file is discarded.

5.3 PPCD Integrity

Protecting PPCD Workspace (WS). In the previous sections we described how individual content-parts and related processes are managed and protected. However, the WS itself also needs to be protected. The PPCD HE creates and controls a WS for each PPCD access session, locks access, giving the HE process an exclusive access to the WS contents. The key challenge here is to enforce access to the workspace by HE process and its children only. This can be achieved:

1. By using an exclusive access lock [13] to the WS folder, which can require a file system driver in Microsoft Windows, and
2. By ensuring the WS folder only exists during the lifecycle of the HE process - automatically discarding it on exit.

Any attempt to read/edit the contents of the WS, copy or move files to/from the WS done by a process other than PPCD will be blocked by the file system driver directory lock.

To ensure integrity of a composite document, the HE needs to modify the behavior of native applications: 1) to disable *Save*/*SaveAs* functionality in View mode to prevent any changes to a RO content-part; 2) to extend *Save*/*SaveAs* functionality in Edit mode to ensure any modifications to a RW content-part are committed back to the PPCD composition. We now describe different techniques that can assist in this task.

Automation: Embedding Native Applications. OLE Automation ([8], [9]), the technology developed by Microsoft, allows the embedding objects into other objects as well as control of their functionality. To provide the required control over *Save/SaveAs* functionality a native document-handling application is invoked in embedded mode with the PPCD HE as its parent process. Fig. 6, left shows the snapshot of different applications (Microsoft Excel, Word, Adobe pdf, etc.) embedded into PPCD Authoring Environment. This solution provides the most unified view of a PPCD composition; however it is limited to only the number of applications supporting OLE Automation.

This solution implies that a main application with multiple embedded applications can result in a relatively big load, which could result in some performance delays on restricted capability devices; and a large GUI, which could not be displayed on small or low resolution screens. Performance issues may be solved using the 'lazy' approach, discussed in Section 5.2.

Add-Ons. Some applications offer an opportunity for customization through Add-Ons or Add-Ins. A good example is Microsoft Office Suite.

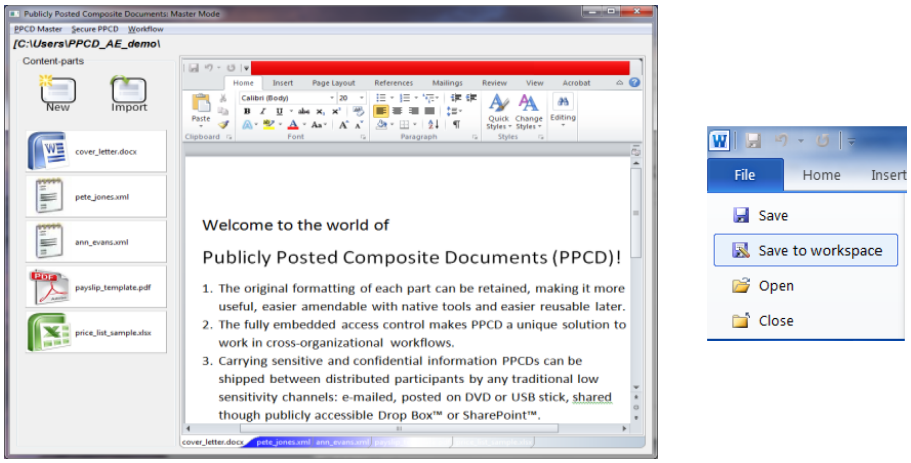

Fig. 6. (left) PPCD document in Authoring Environment: adding documents and materials for a workflow. Content-parts formatted as Microsoft Word, Excel, Adobe PDF and XML are all embedded as OLE objects into the main GUI. (right) PPCD document handled by a native application with augmented save functionality to guarantee integrity.

In the PPCD HE context, Add-Ons are utilized to augment the functionality of native applications and make them comply with the logic flow described in Section 5.2. For example, Fig. 6 right shows the *SaveAs* functionality being replaced by (temporarily) saving into the current Workspace without committing to the PPCD composition. The final changes are committed on the session termination, prompting the user to accept or reject an updated version of each RW part. The original *Save* functionality (Fig. 6, right) was also extended: the document is automatically saved into the Workspace – functionality provided by the native application, followed by content-part encryption, signing and insertion into the original PPCD composition. The native application can be closed at this point and the corresponding file in the Workspace discarded. From the usability point of view this mode of operation could be preferential when the user has finalized his/her changes and wants to de-clutter his Desktop by closing unnecessary applications. It could be a preferential functionality for the lazy approach, discussed in Section 5.2.

The application closure action could also be augmented to warn the user that there are uncommitted modifications, prompting the user to commit or discard them.

If, according to the access granted, a content-part needs to be handled in RO mode, then the preferred solution is to open the native application in View mode. There are 3 types of applications:

Type 1: both Editing and Viewing modes are provided within the same application and can be selected by startup parameters;

Type 2: Editing and Viewing modes are provided by different applications, e.g. Adobe Acrobat for editing pdf-documents and Adobe Reader for viewing;

Type 3: Viewing mode is not supported.

Handling the first two types of applications is very straightforward. Type 2 applications provide yet another justification for the PPCD HEs to maintain their own list of registered native applications as discussed in Section 5.1 where different applications are initialized for the same file extension depending on the access granted.

The situation with type 3 applications is trickier. If a native application offers no View mode, then even after *Save/SaveAs* functionality is being disabled in Edit view, a document is likely to remain active and accept changes. This could be misleading to the user, who could assume RW access to the content-part and proceed with editing, without being able to save. To avoid any confusion or frustration, at the startup point of any such applications, the user is warned that any changes cannot be saved/or cannot be saved into the original composition due to RO access granted. The required Add-Ons disable saving functionality and issue the corresponding warning.

For some type 3 applications, conversion of a RO content-part to Adobe PDF could be performed. The converted part is then presented by Adobe Reader. However, this is a very narrow option: only applicable for applications where all required functionality is supported in PDF.

In some cases, a workflow participant may be allowed to export content-parts out of a PPCD composition, e.g. low sensitivity content-parts where a user is trusted with the contents, etc. In this case, *SaveAs* functionality does not need to be disabled, but renamed to *Export* from the PPCD HE and augmented to warn a user that an exported part is no longer synchronized with the original PPCD composition. Even if the *Export* functionality is allowed for a content–part with RO access granted, *Save* func-

tionality remains disabled as an updated content-part cannot be committed into the original PPCD-composition.

This is a light and reliable solution; however it can only be used for native applications supporting suitable Add-Ons. To summarize, *Save*/*SaveAs* functionality in a native application needs to be augmented and clarified. It can be renamed as *Export* from the PPCD HE for non-sensitive content-parts, and removed for sensitive ones. *Save* to Workspace functionality can be used to replace *SaveAs* or added independently to provide for usability and/or version control. Clear messages need to be shown to avoid potential confusion with modified functionality of otherwise familiar applications.

System Call Interception. While the first two techniques are likely to provide for the majority of office document handling applications, there are still many document applications supporting neither of the techniques. In this section we describe a generic method to capture and control the native application behavior by intercepting the family of system calls responsible for the action of interest, i.e. system calls issued by a native application, when requesting the OS kernel to save a document.

Whilst challenging for the implementation, this technique can potentially support any application and document format.

SCI Solution. Based on the System Call Interception Framework (introduced in [10]), this solution provides a unique way for monitoring and controlling native applications' *Save*/*SaveAs* events. Let us recall that system calls provide the essential interface between applications, running in the user mode, and the kernel mode of the OS. Using the system call interface a program/process/application from the user mode can request a service from the OS kernel. Such a service can be a privileged operation provided by the OS kernel: accessing a hard drive, creating and executing new processes, scheduling and others.

System calls have been used for a long time in a passive way to identify behavioral patterns of user mode applications for audit proposes or for malware analysis. It was recently shown in [10] that the system call interception with application injection can be used more actively to address security challenges such as sensitive data leak prevention, just-in-time document classification and others. In this paper we extend the technique even further to the run-time augmentation of undesired behaviors of native applications within the PPCD HE framework.

When a PPCD HE process creates an instance of a native application (new process) corresponding to the content-part format, a Dynamically Linked Library (DLL) – modifying the behavior of the native application – is injected into the application process. The aim of this DLL is to capture the relevant system calls made by the application to the OS kernel and alter their execution. All I/O system calls issued by a controlled native application that can potentially write data to the disk are captured and detoured to a personalized trampoline function. The system calls captured in Windows are: *WriteFile/Ex/Gather*, *CopyFile/2/Ex/Transacted*, *ReplaceFile*, *FlushFileBuffers* and *MoveFile/Transacted/WithProgress*, in their ANSI and Unicode versions if applicable. Microsoft Detours [15] is the library that has been used in the current prototype, and provides a good example of this behavior modification.

If an application is running in View mode (RO access), the trampoline function simply blocks any write functionality call, hence preventing any save action performed by the application. We need to notice that in this mode of operation, the application might not be able to save any of its configuration/running parameters to the disk, which could cause some undesirable effects for the application.

As this is a generic solution, i.e. extendable to all the applications, any pattern of system calls that need to be detected to improve the usability of a particular application can be captured and analyzed. For example, an application specific extension can be used to allow system calls saving data to a predefined configuration directory. To prevent this behavior from being exploited by malware, the PPCD HEs can take control over the allowed directories.

If an application is running in Edit mode (RW access), the trampoline function analyzes the destination through the meta-data of the system calls. All write calls destined to the dedicated session Workspace are allowed; the trampoline functions execute the original system calls that were intercepted. All write calls to any other destination are blocked by the trampoline function, hence preventing the application from saving data outside of the Workspace.

Unfortunately, the technique does not allow changing any buttons/labels from the native application GUI; however corresponding pop-up messages can be issued to guide and inform users.

SCI: Advantages and Disadvantages. The augmented behavior provided through SCI technique is the most generic and is able to support any native application and extended to any OS, preventing any undesired functionality that could potentially compromise the PPCD integrity.

There are only a limited number of system calls in every OS, and only a small subset of them are responsible for a particular privileged action, such as writing to a hard drive. Capturing and detouring the group of system calls from a native application can effectively augment/prevent the undesired behavior of this application. However, this solution also has its weaknesses. For example, SCI is unable to modify native application GUI or to prevent/suppress some messages produced by native applications. Subsequently, SCI modifications on some untested document applications may result in undesirable user experiences, such confusing messages displayed by an application.

In addition, many applications need to perform disk writes to other locations for other purposes, which by default will be blocked, and hence its functionality may be limited. This can be improved by developing application specific customizations for native applications with different behaviors and handling application specific system call patterns more efficiently, whilst having a generic restrictive default version for applications that have not been analyzed.

SCI technique could be used very efficiently in a so called advisory mode (for non-sensitive contents), when a user is just warned that a content-part is about to be exported out of the PPCD HE and the modified version will not be automatically synchronized with the current workflow version.

5.4 Extension to Other Operating Systems

The prototype has been implemented in Windows, with a special focus on Microsoft Office suite, because they are the most commonly used operating system and document management software, but this solution can be extended to any OS and software.

While OLE is only available in Windows, add-ons are available in multiple software suits, like Apache Open Office. As all the modern operating systems (OS) implement a System Call interface to isolate user and kernel modes, the SCI model can be extended to any other OS, and it can be used to capture the calls made by any user level software to the kernel.

Due to the ever-increased use of mobile devices, Android OS could be a good target to extend the SCI solution, and as it is based on Linux, which is open source, it is much easier to implement SCI than in Windows. These extensions are out of the scope of this paper, but future work in this direction is planned if there is demand for it.

6 Current Prototype and Implementation

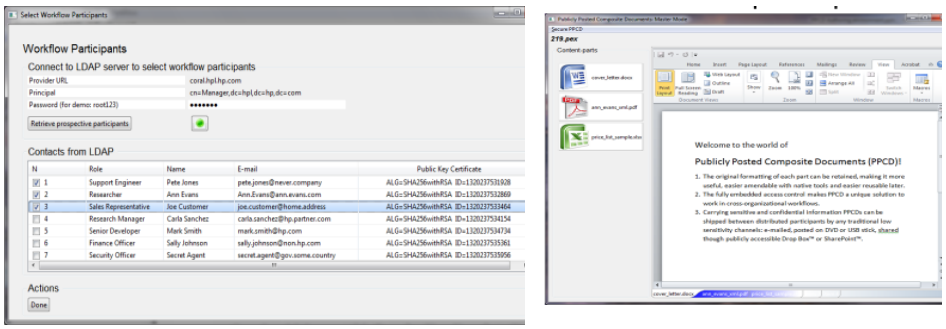

Fig. 7. (left) Assigning workflow participants/roles: known public keys for participants are retrieved from local/central LDAP repositories (Contact book); (right) View of the same document as in fig. as it is presented to one of the workflow participants (according to access granted) in a Workflow Participant environment.

The first prototype of PPCD Authoring Environment (Fig. 6 left and Fig. 7 left) and Workflow Participant Access Environment (Fig. 6 right and Fig. 7 right) were successfully built using Java for the main application GUI, embedded applications, Workspace management and access to the PPCD files. Add-Ons were prototyped in C#. SCI was prototyped using C++, and Microsoft Detours [15] library; the kernel callback *PsSetCreateProcessNotifyRoutine* was used in order to identify new process at the early creation point. Whilst efficient, a kernel callback can be avoided in the future releases of the PPCD HE as all document handling applications are launched by the main PPCD process, thus identifying their creation points is no longer a challenge.

The native applications (Microsoft Word, PowerPoint, Adobe Reader, etc.) were embedded into the main framework using Java/SWT OLE Automation, providing a coherent and intuitive GUI. Snapshots of the PPCD Authoring Environment are shown in Fig. 6 and Workflow Participant Access in Fig. 7 left. The immediate drawback of the PPCD HE working with embedded applications was the slow loading of the GUI (caused by relatively slow loading of Microsoft Office).

Add-Ons are the most desirable solutions from the user experience point of view; hence, system call interception is only reserved to applications that cannot be handled by the other techniques. The combination of the three techniques completely guarantees the integrity of the PPCD, providing a good user experience for the most used applications whilst also protecting any document handled by any unknown present or future native application.

7 Conclusions and Future Work

In this paper, we have investigated the main principles and requirements for creating PPCD Handling Environments: Authoring Environment and Workflow Participant Access. The first prototype of a PPCD HE was successfully built and is now fully operational. Reliable and secure data exchange between native applications and PPCD HEs was one of the key challenges; this was successfully resolved by carefully managed session-based Workspaces. The other most important challenge addressed was augmenting *Save*/*SaveAs* functionality of native applications to prevent uncontrollable content-part export out of the PPCD HE, potentially resulting in the latest modifications not being synchronized back into the original PPCD composition and/or sensitive information exposure.

In the future, we are planning to extend PPCD HEs to other Operating Systems, especially Android mobile platform. We anticipate that the system call interception module will be less complex, because Android is based on Linux, which is an open source platform; hence the code of the kernel is available and can be modified. As the applications are written in Java, another path that may be explored is to implement a middleware between the applications and the Dalvik JVM[17], or exploiting the fact that the applications are sandboxed.

References

[1] H. Balinsky, S. Simske, Differential Access for Publicly-Posted Composite Documents with Multiple Workflow Participants. In Proc. of the 10th ACM Symposium on Document Engineering (DocEng), pages 115-124, September 21-21, 2010.

[2] MSDN, How to: Programmatically Open Existing Documents http://msdn.microsoft.com/en-us/library/tcyt0y1f.aspx, retrieved April, 2014.

[3] Symantec File Share Encryption, http://www.symantec.com/file-share-encryption, http://www.symantec.com/business/support/index?page=content&id=TECH197084, retrieved October 2014.

[4] H. Balinsky, L. Chen, S. Simske, Publicly Posted Composite Documents with Identity Based Encryption. In Proc. of the 11th ACM Symposium on Document Engineering (DocEng), pages 239-248, September 21-21, 2010.

[5] H. Balinsky, S. Simske, L. Chen, Premature silent workflow termination in Publicly Posted Composite Documents, In Proc. of 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pages 1292 – 1297, October 9-12, 2011.

[6] H. Balinsky, L. Chen, S. Simske, Publicly Posted Composite Documents in Variably Ordered Workflows, In Proc. of 2011 IEEE International Conference on Trust (TrustCom), Security and Privacy in Computing and Communications, pages 631-638, November 16-18, 2011.

[7] SQLite, homepage, https://sqlite.org/, retrieved April, 2014.

[8] Object Linking and Embedding http://en.wikipedia.org/wiki/Object_Linking_and_Embedding, retrieved April, 2014.

[9] OLE Automation http://en.wikipedia.org/wiki/OLE_Automation, retrieved April, 2014.

[10] H. Balinsky, D. Perez, S. Simske, System Call Interception Framework for Data Leak Prevention, In Proc. of 2011 IEEE Enterprise Distributed Object Computing Conference (EDOC), 2011, pages 139 - 148, August 29-September 2, 2011.

[11] *ProcessMaker* workflow management software, http://www.processmaker.com/, retrieved April, 2014.

[12] L. Vogel, Microsoft and Java Integration with Eclipse – Tutorial (Object Linking and Embedding), http://www.vogella.com/tutorials/EclipseMicrosoftIntegration/article.html#microsoftole, retrieved April, 2014.

[13] MSDN, LockFileEx function, http://msdn.microsoft.com/en-us/library/aa365203%28v=vs.85%29.aspx, retrieved April, 2014.

[14] HP Protect Tools, FileSanitizer, http://h20331.www2.hp.com/hpsub/cache/281822-0-0-225-121.html, retrieved April, 2014.

[15] Microsoft Detours Express 3.0 http://research.microsoft.com/en-us/projects/detours/, retrieved April, 2014.

[16] MSDN, Kernel-mode driver architecture, driver support routines: *PsSetCreateProcessNotifyRoutine*, http://msdn.microsoft.com/enus/library/windows/hardware/ff559951(v=vs.85).aspx, retrieved April, 2014.

[17] D. Bornstein, Dalvik VM Internals, https://sites.google.com/site/io/dalvik-vm-internals/, retrieved October, 2014.